UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
[GRADUADA/O EN TITULACIÓN]

Driver Linux con soporte Industrial I/O para sensor de consumo

Industrial I/O Linux driver for power monitor

Realizado por
Agustín Téllez Chica

Tutorizado por
Andrés Rodríguez Moreno

Departamento
Arquitectura de Computadores

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2020

Fecha defensa: Julio de 2020

Resumen

La plataforma Raspberry Pi aúna el bajo consumo, bajo coste y la potencia de un SO Linux en una pequeña placa de desarrollo que ofrece múltiples GPIOs (E/S de propósito general) para conectar dispositivos externos con facilidad. En este trabajo se desarrollará un driver cliente i2c para conectar varios sensores de consumo INA219, además el driver ofrecerá las lecturas de los diferentes valores que mide el sensor en el espacio del usuario a través del subsistema del kernel IIO (Industrial I/O) que permite estandarizar la lectura de sensores tipo ADC. Se proporciona los mecanismos para la carga del driver automáticamente y de forma manual. El driver será operativo para cualquier sistema Linux y se validará en la plataforma Rpi.

Palabras clave:

Driver, Linux, Raspberry Pi, Raspbian, dispositivo, IIO, device tree, overlay, Sensor consumo , Ina219.

Abstract

The Raspberry Pi platform combines the low power, low cost, and power of a Linux OS into a small development board that offers multiple GPIOs (general purpose I/O) to connect external devices with ease. In this work an i2c client driver will be developed to connect several INA219 consumer sensors, in addition the driver will offer readings of the different values measured by the sensor in the user space through the kernel subsystem IIO (Industrial I/O) that allows to standardize the reading of sensors type ADC. The driver will be operational for any Linux system and will be validated on the Rpi platform.

Keywords:

Driver, Linux, Raspherry Pi, Rasphian, device, IIO, device tree, overlay , power consum, ina219

Índice

1

Introducción

1.1 Motivación

El control de consumo energético cada vez toma una mayor importancia por diferentes motivos, control de gastos, por su integración en dispositivos móviles que funcionan con baterías y por tanto deben optimizar su consumo. Los dispositivos IoT cada vez más presentes con la llegada de las redes 5G son un ejemplo. Por este motivo necesitamos herramientas para monitorizar y conocer los patrones de consumo. Es necesario conocer dicha información para optimizar las decisiones que afectan a la energía en los sistemas. El dispositivo INA219 es un pequeño dispositivo que podemos usar para el control de consumo en diferentes aplicaciones como servidores, computadoras portátiles, sistemas de control de energía, equipos de telecomunicaciones, cargadores de baterías.

La raspberry Pi se ha convertido en una plataforma para el desarrollo, de formación y aprendizaje ampliamente extendida en universidades y centros educativos por su bajo coste, su versatilidad y potencia. Además de disponer de un sistema operativo Raspbian basado en la distribución Debian, software con licencia GNU presente en multitud se sistemas. El desarrollo en la Raspberry puede ser exportado con facilidad a otros sistemas GNU\Linux.

El desarrollo del controlador mediante el subsistema Industrial I/O nos facilitará el trabajo proporcionando un estándar, unos mecanismos base para la implementación de nuestro controlador. Este subsistema es la base para

implementar todo tipo de controladores que utilicen convertidores de Analógico/Digital o Digital/Analógico como es el caso del dispositivo INA219.

El lenguaje C junto con las herramientas de compilación gcc presentes en Linux serán las bases para el desarrollo de nuestro controlador. Nos obstante, se utilizar otros lenguajes emergentes como Python para el envío de los datos a un servidor de internet para tratar y visualizar la información generada más fácilmente.

1.2 Objetivos

El objetivo principal es crear un controlador basado en el subsistema Industrial I/O para el manejo del dispositivo INA219 que nos pueda servir que referencia para el desarrollo de otros controladores que usen convertidores analógicos/digital o digital/analógico como giroscopios, acelerómetros, sensores de luz y color. El controlador del dispositivo INA219 nos permitirá conocer el Voltaje, la potencia y el consumo en vatios de la entrada conectada al dispositivo INA219.

El controlador debe poder manejar múltiples dispositivos INA219 conectados en el mismo bus I2C. La mayoría de los ejemplos que se pueden encontrar de controladores IIO en internet solo manejan un único dispositivo y un buen controlador debe soportar múltiples instancias de dispositivos para la reutilización del código. No podemos imaginar un controlador USB que solo manejara un único dispositivo. Las configuraciones de estos dispositivos permiten conectar hasta 16 dispositivos en el mismo bus, mediante dos entradas en la que podemos introducir una de las cuatro señales presentes en el bus I2C (GND, Vs, SCL, SDA).

En este proyecto se utilizará la raspberry pi donde cargaremos el modulo del controlador y conectaremos mediante la GPIOs 4 dispositivos INA219 en el bus I2C-1. Utilizaremos una placa de desarrollo para poner los dispositivos y conectarlos. Utilizaremos varios potenciómetros para variar el voltaje de entrada entre 0V y 3,2V girando manualmente y varias luces led con una resistencia para no sobrepasar el rango de funcionamiento de los leds que es inferior, estando en torno a los 2V.

1.3 Estructura de la memoria

La estructura de la memoria se compone de los siguientes capítulos:

1. Introducción

En este capítulo se hará una introducción, un apartado con la motivación, las tecnologías utilizadas y la estructura de la memoria.

2. El segundo capítulo se dedica al dispositivo INA219 donde se especifica que es un INA219 y sus características técnicas.

3. Un capitulo donde se detalla que es el subsistema Industrial IIO, como se define un dispositivo IIO y la definición de canales.

4. El desarrollo del controlador INA219, lenguaje y las herramientas utilizadas. Una primera implementación sin IIO y una segunda implementación utilizando IIO para poder comparar ambas versiones.

5. Carga automática del módulo del controlador IIO en la Raspberry Pi y su utilización. En este capítulo se describe como cargar automáticamente el modulo del controlador en la Raspberry Pi mediante device tree Overlays y el SYSTEMD.

6.En este capítulo se describe como enviar los datos leídos de diferentes canales de los dispositivos INA219 y enviarlos a un panel grafico de un servidor de internet proporcionado por Adrafruit IO en su versión gratuita.

7.Un último capítulo para las conclusiones y posibles líneas futuras.

Un apartado para las referencias bibliografías y de internet.

Un anexo de cómo preparar el entorno de la Raspberry Pi, donde se detalla instalación del sistema operativo, configuración de la Raspberry, configuración del bus i2c, añadir cabeceras del kernel para poder compilar el modulo del driver, librerías de Python.

El anexo B describe las funcionalidades que se han incorporado al modulo de la pantalla oled 1306 utilizada en el proyecto. Se detalla la utilización de la función ioctrl que ofrece la posibilidad de añadir funciones que no son las típicas de un fichero (lectura, escritura), como por ejemplo cambiar el contraste o la fuente utilizada.

1.4 Tecnologías y programas utilizados

- Raspberry Pi: Es un pequeño computador caracterizado por su bajo precio, pero lo suficiente potente y versátil para desarrollo múltiples proyectos. Se ha generalizado en los entornos universitarios y educativos.
  Destaca por su compatibilidad los sistemas GNU/Linux y su distribución Rasphian basada en Debian.
  Tiene un amplio conjunto de conectores y puertos, Ethernet a 1GB, Wifi, USBs , GPIOs, HDMI

- Device Tree Overlays: Los ofrece la posibilidad de añadir dispositivos generalmente opcionales al sistema.

- Git: Es un sistema para el control de versiones software.

- Trello: Es un software que nos permite mantener un control de nuestro proyecto con paneles para definir tareas y estado fácilmente.

- Editor Geany y Notepad++: Editor de texto para Linux y Windows.

- Putty: Software para realizar conexiones SSH a la raspberry Pi. Aunque dispone de HDMI y USB lo normal es utilizar el PC para manejar la raspberry remotamente mediante una conexión SSH.

- Lenguaje C: Es uno de los lenguajes más utilizado en la actualidad y esencial para la construcción de modulo del controlador para Linux.

- Python: Lenguaje que utilizaremos para conectar la raspberry con un servidor web.

- Adrafuit IO: Es un servicio web que nos proporciona unos paneles gráficos para mostrar la información mediante un conjunto de rutinas en Python. Nos permite crear una cuenta gratuita con restricciones de datos, pero suficientes para nuestro propósito.

2

El dispositivo INA219

3.1 El bus I2C.

El bus I2C fue creado en los años 80 por Philips para comunicar diferentes componentes dentro de un sistema electrónico (inter integrated circuits) y se convirtió en un estándar de la industria. Es un bus bidireccional con protocolo maestro/esclavo con comunicación de datos en serie y síncrona.

Se ha ido mejorando pero sigue manteniendo la idea básica:

- Protocolo de dos hilos, uno para transmitir datos (SDA), una línea de reloj (SCL) para sincronizar. Mas una línea de GND y VCC.
- Cada dispositivo tiene una dirección de 7 bits, por tanto podemos conectar hasta 127 dispositivos en el mismo bus i2c.
- Uno dispositivo actúa como master, controla el reloj.
- No se requiere una velocidad porque el master controla el reloj.

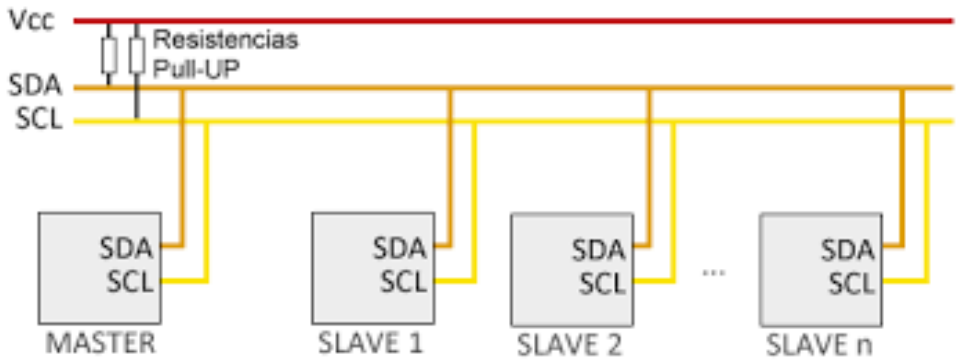


Figura 3: Esquema de conexión bus I2C.

3.2 ¿Qué es un INA219?

El dispositivo INA219 es un sensor diseñado por Texas Instruments [2] que permite medir la tensión, la intensidad y la potencia en un circuito electrónico. Define un interfaz compatible con I2C/SMBUS, líneas VCC, GND, SDA, SLC.

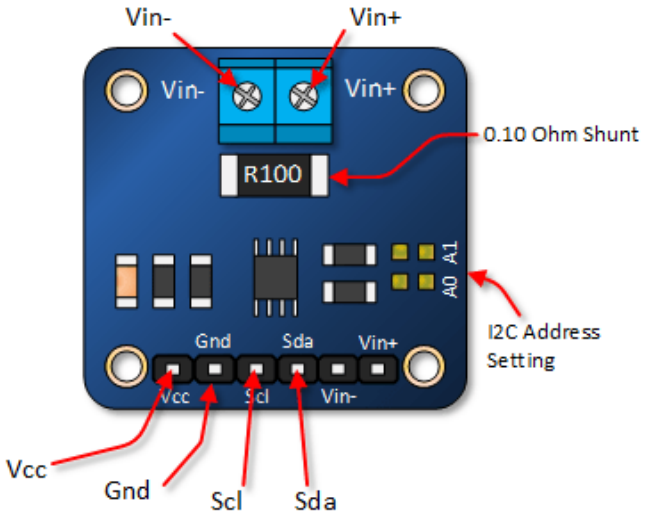


Figura 4: Dispositivo INA219.

Puede medir tensiones entre OV a 26V, configurable en dos escalas de 16V y 32V con una presión máxima de ±0.5.

Se puede calibra por software la ganancia, lo que influye en el rango y precisión. Al mínimo se puede medir corrientes de ±3.2V y al máximo un rango de ±400mA con presión de 0.1mA.

Podemos conectar hasta 16 dispositivos en un mismo bus puesto que dispone de entradas en las cuales podemos introducir una de las cuatro señales presentes en el bus VCC, GND, SDA, SLC con lo cual hay 2^4 opciones para el direccionamiento de INA219.

A1	A0	SLAVE ADDRESS
GND	GND	1000000
GND	V_{S+}	1000001
GND	SDA	1000010
GND	SCL	1000011
V_{S+}	GND	1000100
V_{S+}	V_{S+}	1000101
V_{S+}	SDA	1000110
V_{S+}	SCL	1000111
SDA	GND	1001000
SDA	V_{S+}	1001001
SDA	SDA	1001010
SDA	SCL	1001011
SCL	GND	1001100
SCL	V_{S+}	1001101
SCL	SDA	1001110
SCL	SCL	1001111

Tabla 1. Pines y direcciones de dispositivo INA219.

3.3 Funcionamiento del INA219

Se compone de una resistencia shunt, un amplificado de ganancia y un conversor de analógico a digital de 12bits (ADC).

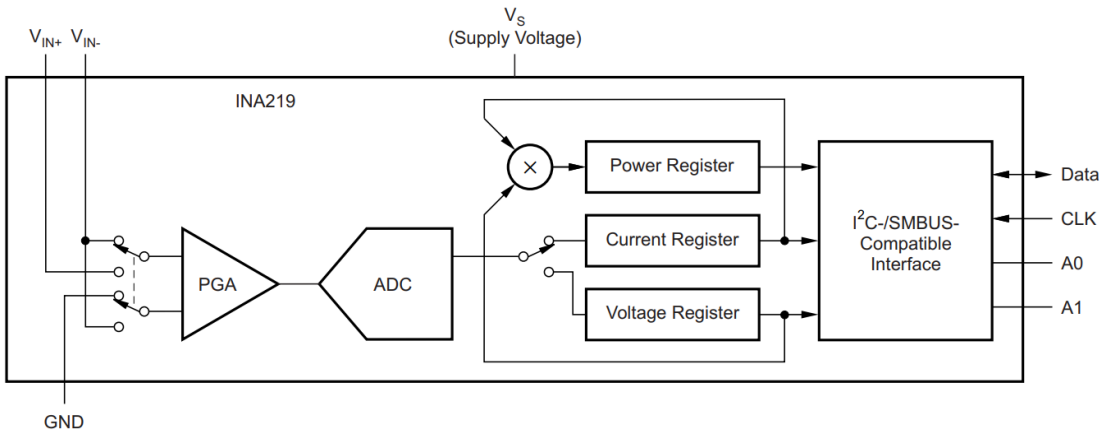


Figura 2: Esquema simplificado del INA219.(www.ti.com data sheet INA219).

EL ADC mide la tanto la tensión del bus como la de la alimentación de carga y puede calcular la potencia eléctrica consumida por la carga mediante la multiplicación de la tensión por la intensidad, P=V*I. El modulo incorpora por defecto una resistencia de 0.1 Ohm junto con un ADC de 12bits proporciona un rango máximo de ±400mA con presión de 0.1mA.

La tensión máxima que puede medirse en la resistencia de shunt es de 40mV. Por defecto, el módulo lleva incorpora una resistencia de 0.1 Ohm. Esto, combinado con el ADC de 12 bits, da lugar a un rango máximo de ±400mA, con una precisión de 0.1mA.

Sin embargo, podemos modificar la ganancia del PGA entre 1 a 8. Esto da lugar a un rango máximo de medición de intensidad de ±3.2A, con una precisión de 0.8mA.

La medición de tensión tiene dos escalas de medición, de 16 y 32V, proporcionando una precisión de hasta 0.5% respecto al máximo de la escala elegida.

3.4 Esquema de conexión con la raspberry Pi mediante bus I2C.

La Raspberry posee una GPIOs de 40 pines utilizando los pines 3 y 5 para el bus i2C-1.

Siendo sus conexiones las siguientes:

- GND con el GPIO 9
- VCC con el GPIO 1
- SCL con el GPIO 5
- SDA con el GPIO 3

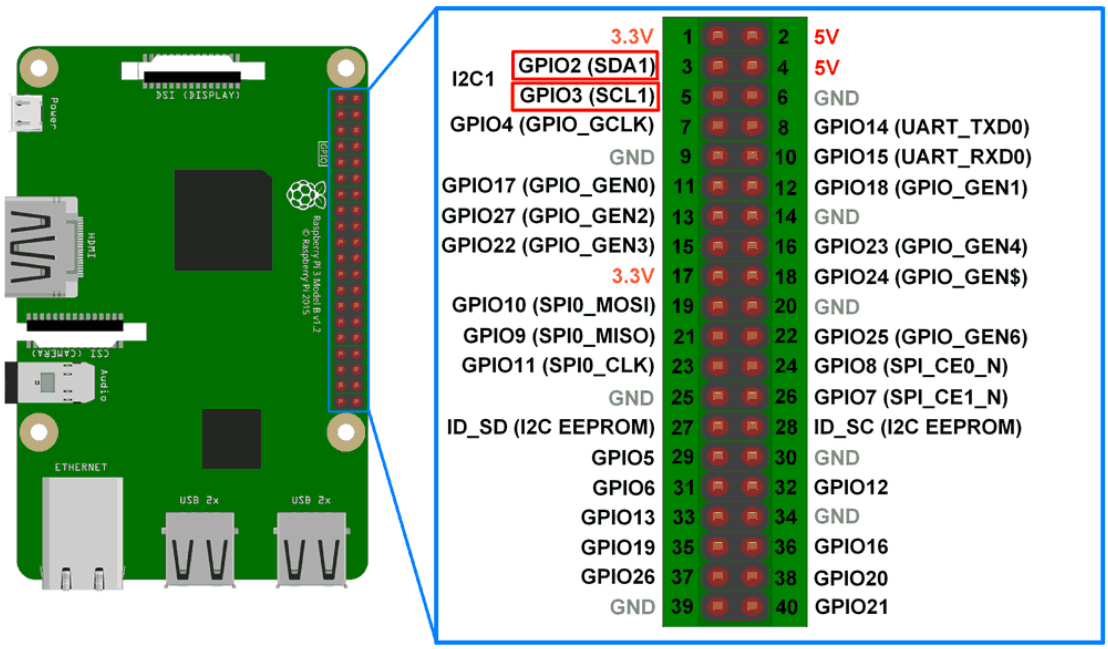


Figura 4. GPIOs de la Rasphberry Pi 3.

3

El subsistema Industrial I/O

2.1 Introducción al subsistema IIO

EL subsistema Industrial I/O [1] está destinado para proporcionar soportes para desarrollar controladores estándar para dispositivos que son conversores de Analógico a Digital (ADC) o Digital a analógico (DAC), entrarían en esta categoría los acelerómetros, giroscopios, sensores de presión, luz y color. Ofrece una interfaz estándar para aplicaciones en el espacio de usuario que manipulan sensores. El objetivo es llenar el vacío entre los subsistemas de monitorización hardware (hwmon) y los de entrada (input) como teclado, ratón, pantallas táctiles.

Un dispositivo IIO típico se conecta a través de un bus I2C o SPI.

2.2 Dispositivos Industrial I/O

El dispositivo IIO corresponde normalmente con un único dispositivo hardware que es manejado por el controlador, el cual proporciona información, integra en un único chip las funciones para el control del dispositivo, normalmente algún tipo de sensor.

En el subsistema IIO se especifica estructura y rutinas para para el manejo de los dispositivos:

iio_dev - Estructura de un dispositivo industrial I/O
iio_device_alloc() - asigna una estructura **iio_dev** desde el controlador
iio_device_free() - libera una estructura **iio_dev** dese el controlador
iio_device_register() - registra un dispositivo en el subsistema IIO
iio_device_unregister() - elimina el registro del dispositivo en el subsistema IIO.

Los modos de operación disponibles para un dispositivo IIO son:

- INDIO_DIRECT_MODE: El dispositivo puede operar con disparadores software.
- INDIO_BUFFER_TRIGGERED: El dispositivo puede operar con disparadores hardware.

- INDIO BUFFER HARDWARE: El dispositivo tiene buffer hardware.

- INDIO ALL BUFFER MODES: La unión de los dos anteriores.

Existen dos formas para que una aplicación de espacio de usuario interactúe con controlador IIO:

Mediante /sys/bus/iio/iio:deviceX que representa a un dispositivo hardware y agrupa todos los canales de datos del mismo dispositivo.

Mediante /dev/iio:deviceX utilizada para la transferencia de datos en buffer y para la recuperación de eventos.

Normalmente un driver IIO se registra el mimo como un controlador I2C O SPI y crea dos rutinas, probe y remove.

Cuando se realiza la llamada a la rutina probe:
1. Se reserva memoria para la para el dispositivo mediante iio_device_alloc().

2.Inicializa la información del dispositivo como el nombre y sus canales.

3.Regista el dispositivo con iio_device_register(). Después de esta llamada el dispositivo esa preparado para aceptar llamadas desde el espacio de usuario.

En la invocación a la rutina remove:

1. Se elimina el registro del dispositivo mediante iio_device_alloc() desde subsistema IIO

2. Se libera la memoria con iio_device_free desde el subsistema IIO.

2.3 El interfaz en el sistema de ficheros de un dispositivo Industrial I/O

En el registro del dispositivo en el sistema se crean unas rutas en el sistema de archivo en /sys/bus/iio/iio:deviceX
que representan el dispositivo y podemos interactuar mediante diferentes atributos y canales de datos.

Entre los atributos comunes están:

name: contiene la descripción de un dispositivo físico.
dev: muestra los valores de minor y major asociados a controlador de dispositivo /dev/iio:deviceX

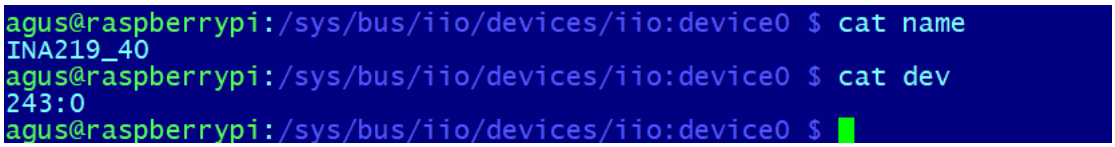


Figura 1: Muestra el nombre del dispositivo y los valores de minor y major asociados al dispositivo INA219 en la dirección 0X40

2.4 Canales de los dispositivos Industrial I/O

Un canal de un controlador IIO representa un canal de datos. Podemos tener uno o múltiples canales para cada dispositivo. Por ejemplo, para un dispositivo INA219 podemos tener un canal para leer el voltaje, otro para leer el consumo y otro para la potencia.

Los canales son especificados mediante la estructura iio_chan_spec.

```
static const struct iio_chan_spec temp_channel[] = {
          {
              .type = IIO_TEMP,
              .info_mask_separate                              =
BIT(IIO_CHAN_INFO_PROCESSED),
          },
     };
```

Se pueden definir diferentes propiedades de los canales mediante mascaras

info_mask_separate, el atributo es para un único canal
info_mask_shared_by_type, el atributo es compartido por todos los canales del mismo tipo.
info_mask_shared_by_dir, el atributo es compartido por todos los canales de una misma dirección.
info_mask_shared_by_all, el atributo es compartido por todos los canales.

```
static const struct iio_chan_spec ina219_channels[] = {
          {
                    .type = IIO_VOLTAGE,
                    .info_mask_separate                        =
BIT(IIO_CHAN_INFO_PROCESSED)
          },
          {
                    .type = IIO_CURRENT,
                    .info_mask_separate                        =
BIT(IIO_CHAN_INFO_PROCESSED)
          },
          {
                    .type = IIO_POWER,
                    .info_mask_separate                        =
BIT(IIO_CHAN_INFO_PROCESSED)
          },
};
```

Encontramos los diferentes tipos de canales predefinidos en el fichero "include/uapi/linux/iio/types.h"

IO_VOLTAGE,
IIO_CURRENT,
IIO_POWER,
IIO_ACCEL,
IIO_ANGL_VEL,
IIO_MAGN,
IIO_LIGHT,
IIO_INTENSITY,
IIO_PROXIMITY,
IIO_TEMP,
IIO_INCLI,
IIO_ROT,
IIO_ANGL,
IIO_TIMESTAMP,
IIO_CAPACITANCE,
IIO_ALTVOLTAGE,
IIO_CCT,
IIO_PRESSURE,
IIO_HUMIDITYRELATIVE,
IIO_ACTIVITY,
IIO_STEPS,
IIO_ENERGY,
IIO_DISTANCE,
IIO_VELOCITY,
IIO_CONCENTRATION,
IIO_RESISTANCE,
IIO_PH,
IIO_UVINDEX,
IIO_ELECTRICALCONDUCTIVITY,
IIO_COUNT,
IIO_INDEX,
IIO_GRAVITY,
IIO_POSITIONRELATIVE,
IIO_PHASE,
IIO_MASSCONCENTRATION,

4

Desarrollo del controlador IIO INA219

4.1 ¿Qué es controlador de dispositivo?

El controlador de dispositivo [3] es un software que permite controlar un dispositivo hardware que se encuentra en el sistema. El controlador se ejecuta en el kernel del sistema operativo con mayor privilegio, permitiendo al sistema operativo o a un programa acceder a sus funciones desde el espacio de usuario con privilegios restringidos.

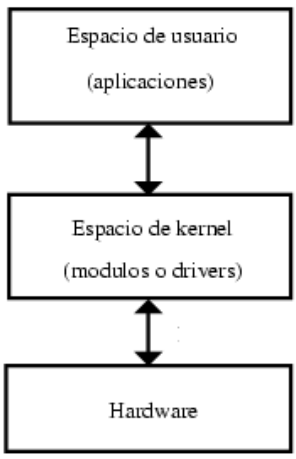


Figura 5: Esquema del espacio usuario /kernel

Los programas que utiliza el usuario final, tales como las "shell" residen en el espacio de usuario ("user space"). Como es lógico estas aplicaciones necesitan interaccionar con el hardware del sistema, pero no lo hacen directamente, sino a través de las funciones que soporta el kernel.

4.2 Las cabeceras del kernel.

El desarrollo de un módulo del kernel requiere tener las cabeceras (headers) para poder compilarlo. Por defecto, estas cabezas del kernel no vienen instaladas en las distribuciones de Rapberry Pi siendo necesario instalarlas desde un Repositorio.

En la Raspberry el comando "sudo apt install raspberry-kernel-headers" instala las cabeceras del kernel.

La versión de las caberas deben coincidir con la versión del kernel instalado. Podemos comprobar la versión del mediante el comando "uname -r".

Una vez instaladas las cabeceras podemos compilar modulos mediante Makefile.

4.3 Primera version del controlador i2c.

El controlador (driver) será un módulo cargable de Linux, para crear un nuevo módulo creaemos un archivo en C que será compilado mediante un archivo Makefile. Los módulos para el kernel de Linux definen las macros init y exit, para su início y finalización.

```
module_init(ina219_i2c_init);
module_exit(ina219_i2c_cleanup);
```

La estructura del modelo de dispositivo se organiza en tres estructuras principales:

- Struct bus_type la cual representa un tipo de bus, en este caso el bus I2C.

- Struct device_drive que representa un controlador de un dispositivos en un bus.

- Struct device representa un dispositivo conectado en un bus.

En la siguiente estructura definimos los manejadores que tendrá el controlador, iniciar, eliminar y la tabla de controladores.

```
MODULE_DEVICE_TABLE(i2c, ina219_i2c_id);

static struct i2c_driver ina219_i2c_driver  = {
     .driver = {
          .owner = THIS_MODULE,
          .name  = "ina219_i2c",
     },

     .probe    = ina219_probe,
     .remove   = ina219_remove,
     .id_table = ina219_i2c_id,

};
```

El último paso es registrar la estructura del controlador en el bus.

```
static int __init  ina219_i2c_init(void){

     return i2c_add_driver(&ina219_i2c_driver);

}
```

Para descargar el driver:

```
static void __exit ina219_i2c_cleanup(void){

     i2c_del_driver(&ina219_i2c_driver);

}
```

Al registar el dispositivo de caracteres se asociará en /dev una estructura file_operations con punteros a cada una de las funciones que soporta el controlador. Cuando el controlador se descarga es necesario deshacer esta asociación para que el nucleo no intente invocar funciones que ya no están disponibles.

La función probe() es llamada cuando existe una coincidencia entre un dispositivo y el controlador que puede manejarlo. Realiza las siguientes funciones:

- Inicializar el controlador. Para la configuración inicial se ha utilizado el ejemplo de la hoja de especificaciones del módulo de Adafruit ina219 [6]. Se han definido tres funciones para realizar diferentes calibraciones del dispositivo:

    1. void setCalibration_32V_2A(struct i2c_client * my_client): Calibra el sensor para ser utilizado para un voltaje máximo de entrada de 32V y una intensidad de corriente máxima de 2 amperios.

    2. void setCalibration_32V_1A(struct i2c_client * my_client): Calibra el sensor para un voltaje máximo de entrada de 32V y una intensidad de corriente máxima de 1 amperio.

    3. void setCalibration_16V_400mA(struct i2c_client * my_client): Calibra el sensor una ser utilizado para un voltaje máximo de entrada de 16V y una intensidad de corriente máxima de 400 miliamperios. Esta es la configuración que nos ofrece la máxima precisión, medición de corriente de 0.1 miliamperio a expensas de admitir un rango menor de voltaje, máximo 16 voltios. Esta es la configuración inicial que utiliza el controlador porque se van a medir valor de voltajes entre 0 y 3.2 voltios. Esta calibración se realiza escribiendo mediante el bus i2c un registro de calibración calculado para esta configuración.

- Preparar el dispositivo para funcionar, reserva las estructuras, reserva memoria, mapea la memoria Entrada/Salida, registra las interrupciones.

- Cuando todo esta listo registra el nuevo dispositivo en el sistema.

Se puede declarar un dispositivo mediante dos formas:

1. Automáticamente en el inicio mediante del Device Tree Overlay :
   Especificando en un archivo las propiedades(nombre, tipo de bus, la dirección en el bus, y el nombre del controlador compatible.

```
/dts-v1/;
/plugin/;

/{
     compatible = "brcm,bcm2709";

     fragment@0 {
          target = <&i2c1>;
          __overlay__ {
               #address-cells = <1>;
               #size-cells = <0>;
               status = "okay";

               ina21941: ina21941@41 {
                         compatible = "ina219_i2c";
                         reg = <0x41>;
                         status = "okay";
               };

               ina21942: ina21942@42 {
                         compatible = "ina219_i2c";
                         reg = <0x42>;
                         status = "okay";
               };



          };
     };

};
```

El archivo creado con el formato .dts se compilar con la herramienta dtc del sistema.

```
dtc -W no-unit_address_vs_reg -O dtb -o ina219i2c.dtbo -b 0-
@ ina219i2c.dts
```

y se copiar al device tree overlay de la raspberry Pi.

```
sudo cp ina219-i2c.dtbo /boot/overlays
```

Para activar este nuevo dispositivo se incluirá en el fichero /boot/config.txt las nuevas líneas a carga:

```
dtoverlay=ina219i2c
```

2. Manualmente mediante la línea de comando:

   Por ejemplo, si se quiere declarar un nuevo dispositivo el en bus i2c-1 en la dirección 0x041 del tipo ina219i2c.

   ```
   echo ina219i2c 0x41 | sudo tee /sys/bus/i2c/devices/i2c-1/new_device
   ```

El controlador pertenece al grupo de controladores de caracteres y por tanto se debe definir las operaciones que va a realizar el controlador. La estructura file_operations nos permite definir cómo va a trabajar el controlador mediante el sistema de archivo.

```
static const struct file_operations iio_fops = {
    .owner     = THIS_MODULE,
    .read = ina219_read,

    //.unlocked_ioctl    = ssd1306_ioctl,
};
```

En este caso definimos la operación de lectura porque el dispositivo es un sensor del cual vamos a leer los valores. No obstante se puede definir operaciones de escritura para manejar el driver desde el sistema archivos y operaciones especiales definidas mediante ioctrl. En el apéndice B se detalla como el uso ioctrl para el controlador de la pantalla oled y como se han añadido funcionalidades al modulo de su controlador mediante ioctrl.

4.4 Compilación del Módulo con Make.

Una vez tenemos completo el código del controlador debe ser compilado mediante un Makefile. Un Makefile es un archivo que nos ofrece una forma simple de organizar la compilación de código:

En siguiente archivo Makefile contiene la información para poder compilar el módulo cargable, añadiendo el path del núcleo de Linux (Kernel) y las instrucciones de cada las operaciones compile, install, uninstall y clean. Cuando usamos el comando make con una de estas opciones se ejecuta la sección correspondiente a la etiqueta del archivo:

-make clean : Elimina los archivos de una compilación anterior.

-make compile: Compila el modulo ejecutando las instrucciones incluidas en la etiqueta compile. Si la compilación tiene éxito se generar un archivo con el nombre del módulo y extensión .ko.

-make install: Carga en el sistema el modulo previamente compilado.

-make uninstall: Descarga del sistema el modulo especificado en el Makefile.

```
MODULE=ina219i2c

KERNEL=`uname -r`
KERNEL_SRC=/lib/modules/${KERNEL}/build

obj-m += ${MODULE}.o

compile:
     @echo Usando kernel ${KERNEL}
     make -C ${KERNEL_SRC} M=${CURDIR} modules

install:
     sudo insmod ${MODULE}.ko
     dmesg | tail
     #echo  ina219i2c  0x41  |  sudo  tee  /sys/class/i2c-
adapter/i2c-1/new_device
     sudo chgrp i2c /dev/ina219i2c41
     sudo chmod go+rw /dev/ina219i2c41
```

```
	sudo chgrp i2c /dev/ina219i2c42
	sudo chmod go+rw /dev/ina219i2c42

uninstall:
	sudo rmmod ${MODULE}
	dmesg | tail

clean:
	rm -f ${MODULE}.ko
	rm -f ${MODULE}.o
	rm -f ${MODULE}.mod.o
	rm -f ${MODULE}.mod.c
	rm -f modules.order
	rm -f Module.symvers
```

Una vez compilado el módulo y obtenemos el archivo .ko podemos probar el módulo utilizando las herramientas insmod y rmmod, para instalar y desinstalar los módulos. Solo el superusuario puede cargar y descargar módulos, por tanto utilizaremos delante de estos comandos la palabra sudo para ejecutar con privilegios elevados.

Podemos ver el resultado de la carga mediante el comando dmesg

La herramienta i2cdetect nos proporciona información del bus i2c y la dirección de los dispositivos presente en el bus.

```
agus@raspberrypi:~/tfg-ina219/driver $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- UU -- -- --
40: UU 41 UU -- UU -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figura 6: Dispositivos en el bus i2c-1

El la figura 7 se muestra la información del bus i2c-1 donde encontramos 5 dispositivos presente en el bus en las direcciones 3c,40,41,42,43 de los cuales 4 están en uso por un controlador y el dispositivo de la dirección 41 no está siendo usado por ningún controlador.

Al registrar un dispositivo i2c en el sistema se crea el interfaz /dev/device por el cual podemos manejar el dispositivo.

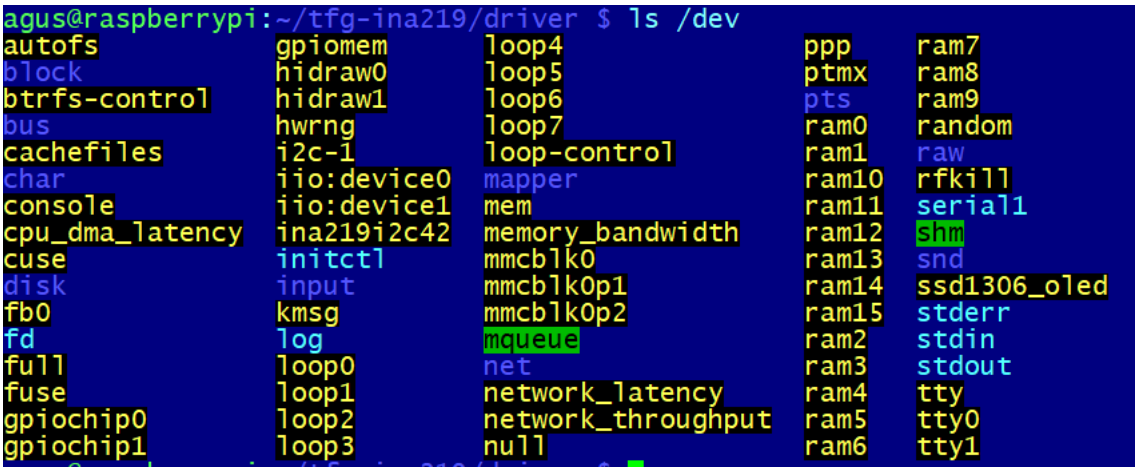


Figura 7: Listado de dispositivos en Raspberry Pi

Mediante estos interfaces el sistema de ficheros podemos manejar el dispositivo asociado.

En este driver i2c se ha definido la función read del controlador para devolver el VOLTAJE desde el espacio del kernel al espacio de usuario.

```
agus@raspberrypi:~/tfg-ina219/driver $ cat /dev/ina219i2c41
BUS VOLTAGE INA219: 2048 mV
BUS VOLTAGE INA219: 2044 mV
BUS VOLTAGE INA219: 2040 mV
```

Figura 8. Lectura del Voltaje mediante controlador i2c.

4.5. Esquema general del funcionamiento.

En este aparatado vamos a realizar un esquema global de funcionamiento del módulo del controlador y su utilización.

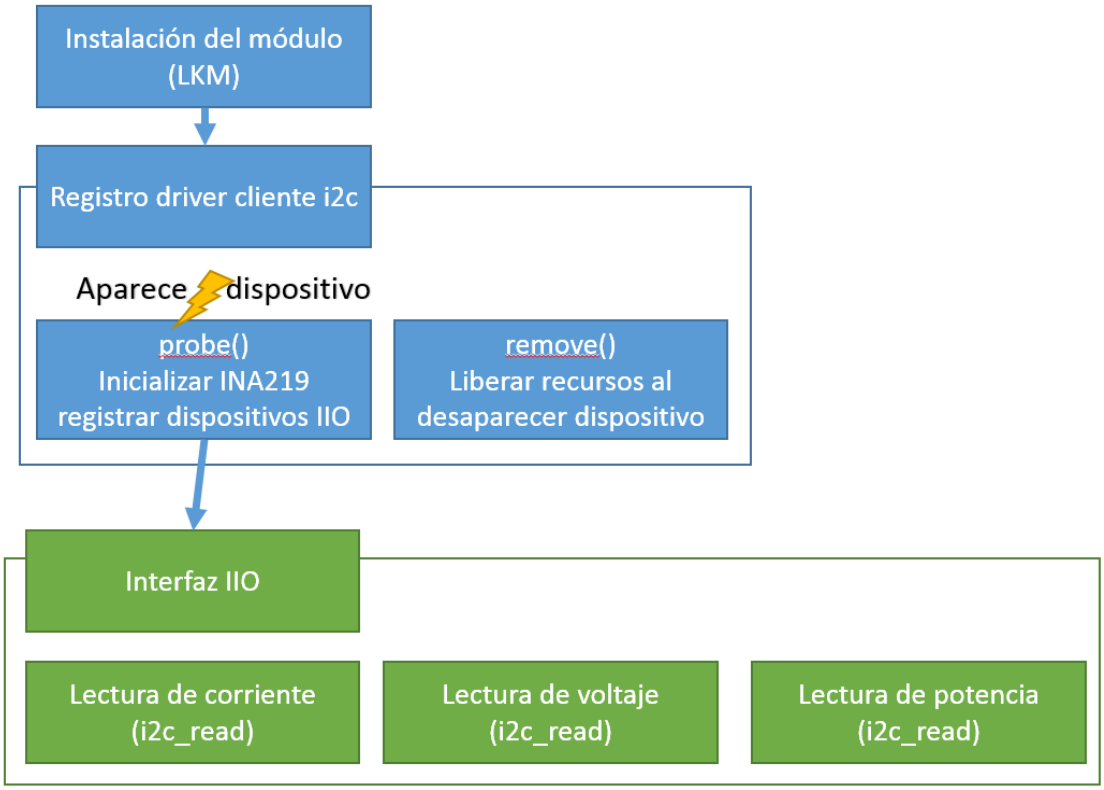


Figura 9: Esquema general de funcionamiento.

El módulo cargable de Linux permite añadir el controlador del dispositivo sin tener que recompilar el núcleo completo. De esta forma podemos agregar soporte al nuevo hardware del ina219, escondiendo los detalles del funcionamiento hardware interno, permitiendo su utilización desde el espacio de usuario a través de un conjunto de llamadas estándar.

En la carga del módulo se registran los clientes i2c que contienen la información de los dispositivos presentes en bus. Una vez está presente el cliente en el bus y hay una coincidencia con un controlador compatible se invoca la función probe que inicializará el dispositivo y lo añade a la lista de dispositivos presentes en el sistema. Al añadirlo al sistema se crean las estructuras y el interfaz para poder utilizar el dispositivo a través del sistema de archivo. De esta forma

se puede invocar las funciones para controlar el dispositivo desde el espacio de usuario.

Al realizar una llamada de lectura del archivo publica como un dispositivo en el sistema fichero el controlador invocará a función de lectura devolviendo en el espacio de usuario los datos leído atraves del cliente i2c del dispositivo asociado.

4.6 El controlador i2c con interfaz Industrial I/O.

El subsistema Industrial I/O nos va a permitir reescribir el controlador para obtener uno más estándar para este tipo de contralores que entran en la categoría de convertidores de Analógico a Digital (ADC) o Digital a Analógico (DAC). Además, nos proporciona una forma más segura y fácil para trabajar con estos dispositivos permitiendo la creación de canales de datos.

Para este controlador INA219 vamos a definir tres canales independientes:

```
static const struct iio_chan_spec ina219_channels[] = {

{
        .type = IIO_VOLTAGE,
        /*Los tipos se encuentran en el fichero "include/uapi/linux/iio/types.h" */
        .info_mask_separate = BIT(IIO_CHAN_INFO_RAW)
        },
        {
        .type = IIO_CURRENT,
        .info_mask_separate = BIT(IIO_CHAN_INFO_RAW)
        },
        {
        .type = IIO_POWER,
        .info_mask_separate = BIT(IIO_CHAN_INFO_RAW)
        },
};
```

Estos canales nos permiten leer la informacion de los regitros del controlador, el voltaje, la intensidad y la potencia. En canales separados por cada dispositivo.

En la función read especificamos como leer en funcion del canal:

```
switch (chan->type) {

        case IIO_VOLTAGE:
                //*val = ret = (((i2c_smbus_read_word_data(data->client,
INA219_REG_BUSVOLTAGE ))>>3)*4);

                *val=ret=getBusVoltage_raw(data->client);
        if (ret < 0)
            dev_err(&data->client->dev,
                "failed to read ADC%d value\n", *val);
        *val2=1000;

                ret = IIO_VAL_INT;

                break;
```

Cuando realizamos una lectura o escritura sobre del dispositivo el driver internamente utiliza las funciones de escritura y lectura sobre el bus i2c mediante el cliente i2c registrado. Las funciones de lectura y escritura utilizadas sobre el bus i2c son:

Envío de un comando atraves de I2C

```
void wireWriteRegister(struct i2c_client * my_client, uint8_t reg, int value)
        {
                uint8_t data[2];
                data[0] = (value >> 8);
                data[1] = (value & 0xFF);
                i2c_smbus_write_i2c_block_data(my_client, reg, 2, data);
        }
```

Lectura de un valor de 16 bits sobre I2C

```
void wireReadRegister(struct i2c_client * my_client, uint8_t reg, int * value)

        uint8_t data[2];
        i2c_smbus_read_i2c_block_data(my_client, reg, 2, data);
        *value  = ((data[0] << 8) | data[1]);
}
```

Utilizando estas dos funciones sobre el cliente i2c y los registros internos del dispositivos podemos definer las operaciones que queremos realizar sobre dispositivo. Por ejemplo, leer el registro que almacena la potencia:

```
int getPower_raw(struct i2c_client * my_client)
{
        int value;
        wireReadRegister(my_client,INA219_REG_POWER, &value);
        return (int) value;
}
```

De esta forma nos facilita la lectura de los valores desde el espacio del kernel a espacio de usuario y la conversion de los valores de caracteres a tipo concreto.

Al cargar un nuevo dispositivo asociado a un controlador IIO el sistema genera un nuevo interfaz el sistema de fichero en /sys/bus/iio/devices/iio:deviceX

```
agus@raspberrypi:/sys/bus/iio/devices $ cd iio:device0
agus@raspberrypi:/sys/bus/iio/devices/iio:device0 $ ls
dev              in_power_raw    name      power      uevent
in_current_raw   in_voltage_raw  of_node   subsystem
agus@raspberrypi:/sys/bus/iio/devices/iio:device0 $
```

Figura 10: device0 mediante controlador IIO

Ahora podemos leer los valores asociado a cada canal, voltaje, intensidad y potencia.

```
agus@raspberrypi:/sys/bus/iio/devices/iio:device0 $ cat in_voltage_raw
1792
agus@raspberrypi:/sys/bus/iio/devices/iio:device0 $ cat in_current_raw
200
agus@raspberrypi:/sys/bus/iio/devices/iio:device0 $ cat in_power_raw
0
```

Figura 11: Lectura de los canales del mediante IIO.

5

Carga automática de módulo del driver

En este capítulo vamos a definir los pasos necesarios para cargar automáticamente el controlador de dispositivo IIO en la raspberry Pi.

5.1 Device Tree Overlays

La raspberry Pi utiliza un Device Tree (DT) [5] para describir el hardware presente en el sistema. Este Device Tree incluye los parámetros que proporcionan el control sobre las carácteristicas presentes en la placa.

El cargador del firmware es el encargado de cargar el Device Tree.

En el Device Tree Overlays podemos definir hardware que puede ser opcional en el sistema.

El cargador soporta directivas mediante el fichero config.txt:

```
dtoverlays=ina219i2c
dtoverlays=ssd1306_oled
```

5.2 SYSTEMD

El systemd es utilizado en en los sistemas de inicio de Linux (El proceso init llamado por el kernel para iniciar el espacio de usuario durante el arranque de linux y posteriormente para gestionar todos los demás procesos). Se diseñó para el núcleo de Linux y tiene como objetivo unificar configuraciones básicas y comportamientos de los servicios en las diferentes configuraciones.

En este sistemd podemos configurar un servicio que active el modulo del controlador durante el proceso de arranque del sistema. Para ello se prepara un archivo con la siguiente configuración:

ina219_iio_service:

```
[Unit]
Description=Carga modulo driver iio para el sensor ina219

[Service]
Type=oneshot
ExecStart=/bin/bash                              /home/agus/tfg-
ina219/driver/iio/load.sh
RemainAfterExit=true
ExecStop=/sbin/rmmod ina219iio.ko

[Install]
WantedBy=multi-user.target
```

El archivo se copia en /lib/systemd/system.

El servicio se activa con el comando:
`sudo systemctl enable ina219_iio.service`

5.3 Verificación de la carga del módulo en el arranque.

Con el comando systemctl se puede consultar el estado de los servicios activados en el systemd.

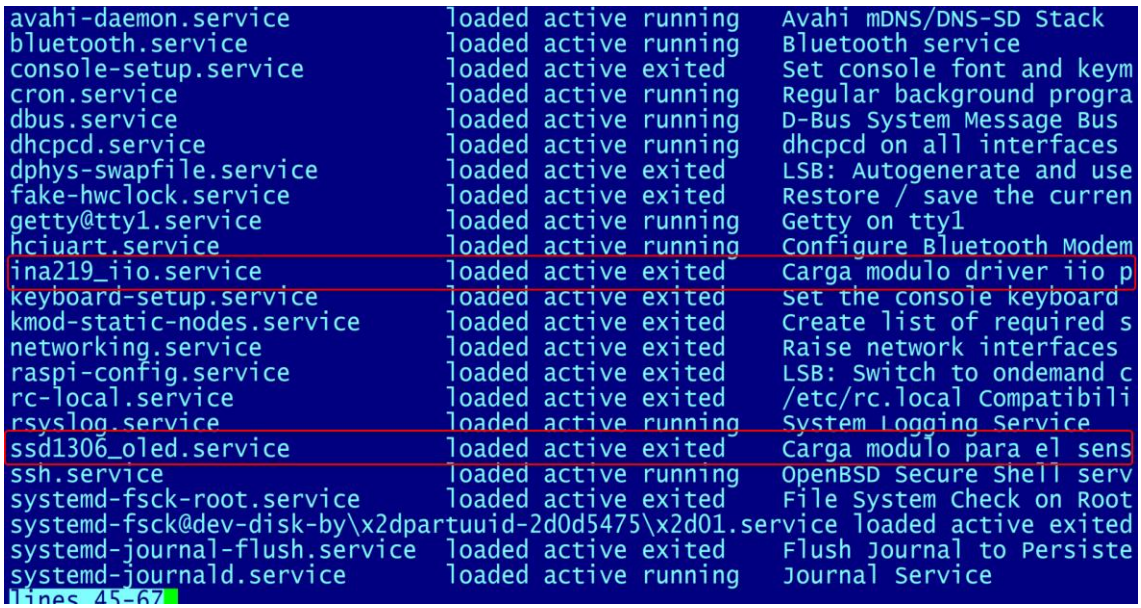


Figura 12: Systemctl

En el systemd se ha cargado un driver i2c para controlar una pequeña pantalla oled ssd1306 para proporcionar información debido a que la Rasberry suele usarse sin pantalla por medio de una consola SSH desde un PC y es de bastante utilidad. Este controloador i2c es un módulo cargable basado el TFG de José Carlos Navarro González (https://riuma.uma.es/xmlui/handle/10630/16907) al que he añadido muchas de las opciones de líneas futuras que dejo sin desarrollar como el rotado de texto en pantalla, video inverso, funciones para definir el contrastes y cambio de fuentes.

Para probar nuestro driver se ha preparado una placa de desarrollo con 4 sensores ina219, una pantalla oled 1306, dos potenciómetros para poder cambiar el voltaje de la alimentación de los dispositivos ina219 y varios leds para ver de manera visual como cambia el voltaje en el circuito. Los leds rojos no pueden pasar los 2 Voltios y por tanto hemos instalado unas resistencias para limitar a esos 2 voltios puesto que la raspberry Pi suministrar 3,2V.

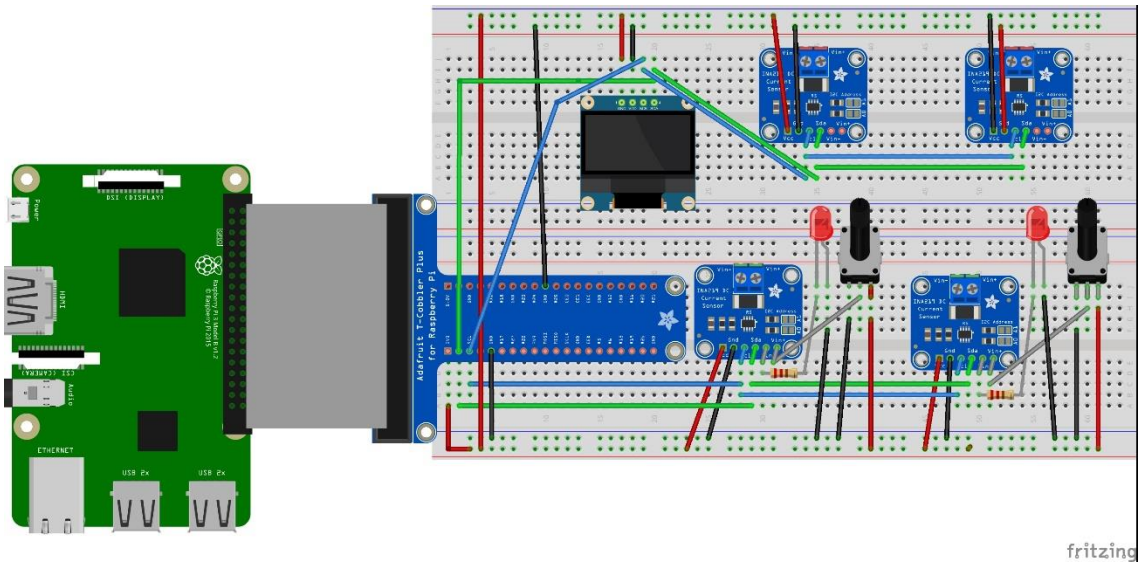


Figura 13: Esquema de conexión protoboard con la raspberry pi.

6

Visualización de resultados

6.1 Envío de datos a Adafruit IO, Panel Grafico.

En este capítulo vamos a utilizar un servicio web que nos ofrece visualizar de forma gráfica y mucho más atractiva la información que leemos del sensor ina219. En el servidor https://io.adafruit.com [7] podemos crear una cuenta gratuita que nos ofrecerá la creación de paneles gráficos (Dashboads), y la fuente de datos (feeds) de la cual se van a mostrar los datos, es este caso los datos leídos del sensor ina219.

Esta cuenta gratuita es totalmente funcional excepto que tiene unas limitaciones en cuanto al número de datos que podemos enviar. Este límite está en 30 datos por minutos que era suficiente para una demostración de la funcionalidad.

El servicio ofrece una completa API con librerías de clientes como Arduino C++, CircuitPython, Phyton, Ruby, Nodel.js, Go. Utilizaremos la librería de Phyton en nuestra Raspberry Pi para enviar los datos leídos desde el sensor.

Al crear la cuenta en el servicio nos ofrecerá un clave AIO la cual nos permitirá conectar la librería con la cuenta del servicio Adrafruit IO. Entrando en la cuenta creada y pulsando el icono de información o desde la barra de menú.

Figura 14: Botón AIO Key

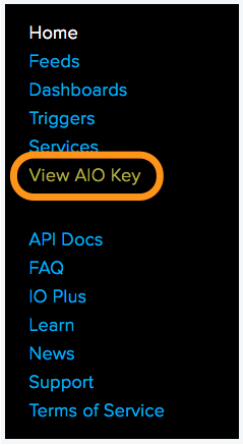


Figura 15: Mostrar la clave AIO.

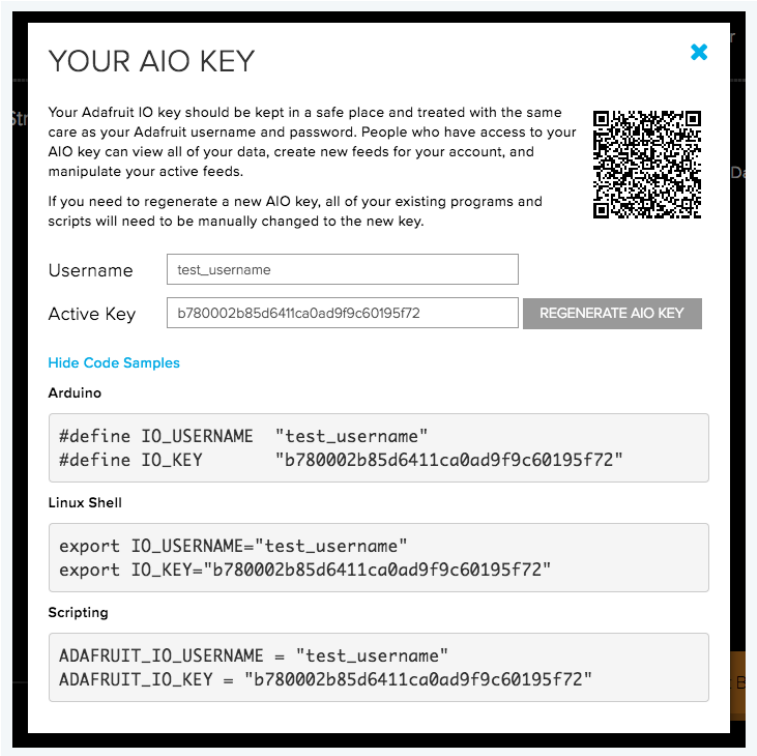


Figura 16: Clave AIO (Ayuda mostrada en la web del servicio io.Adafruit.com.

Las librerías de Phyton para usar el servicio de Adafruit IO se descarga de un repositorio git en la raspberry:

https://github.com/adafruit/Adafruit_IO_Python

En el repositorio se encuentra el software, documentación y ejemplos de utilización del servicio.

Una vez descargado el contenido del repositorio podemos instalarlo mediante el siguiente comando:

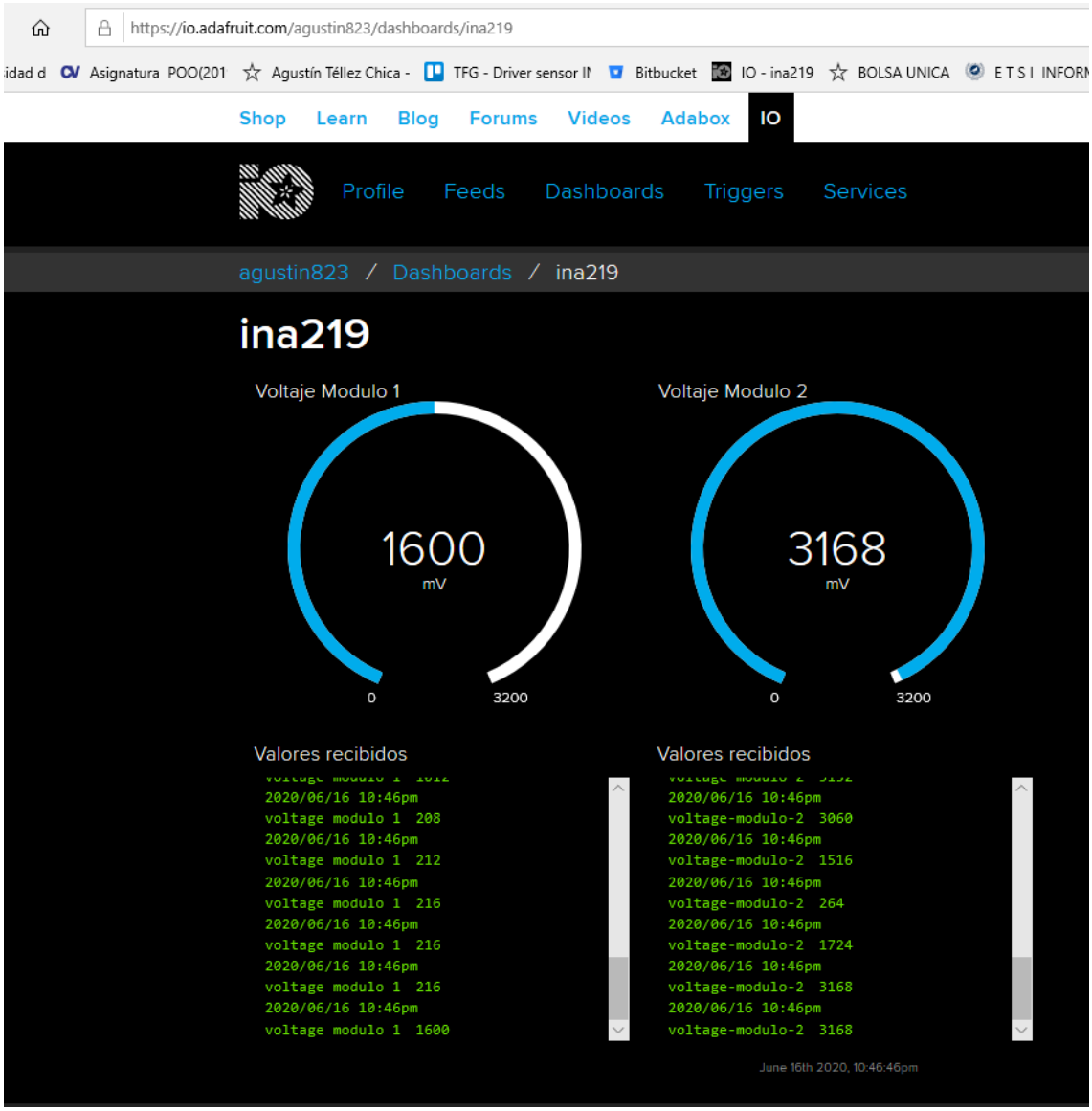
```
python setup.py install
```

Ya tenemos todo preparado para poder enviar los datos al servidor, utilizaremos el usuario y la clave AIO para configurar el ejemplo de Python suministrado en el repositorio.

```
# Simple example of sending and receiving values from
Adafruit IO with the REST
# API client.
# Author: Tony Dicola, Justin Cooper

# Import Adafruit IO REST client.
from Adafruit_IO import Client, Feed, Data, RequestError
import datetime
import time
numero=0

# Set to your Adafruit IO key.
# Remember, your key is a secret,
# so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = 'aio_QdNL50nRK3tbuJKO1DQlOmYCAAVD'

# Set to your Adafruit IO username.
#  (go  to  https://accounts.adafruit.com  to  find  your
username)
ADAFRUIT_IO_USERNAME = 'agustin823'

# Create an instance of the REST client.
aio = Client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)
try:
    voltage1 = aio.feeds('voltage-modulo-1')

except RequestError:
    feed = Feed(name="voltage-modulo-1")
    voltage1 = aio.create_feed(feed)
```

```
try:
    voltage2 = aio.feeds('voltage-modulo-2')

except RequestError:
    feed = Feed(name="voltage-modulo-2")
    voltage2 = aio.create_feed(feed)

#
# Adding data
#
filename                                                    =
"/sys/bus/iio/devices/iio:device1/in_voltage_raw"
filename2                                                   =
"/sys/bus/iio/devices/iio:device0/in_voltage_raw"

# open the file for reading
filehandle = open(filename, 'r')

while numero <= 10:
    # read a single line
    # open the file for reading
    filehandle = open(filename, 'r')
    line = filehandle.read()
    filehandle.close()
    if not line:
        break
    aio.send_data(voltage1.key, line)
    print(line)

    filehandle = open(filename2, 'r')
    line = filehandle.read()
    filehandle.close()
    if not line:
        break
    aio.send_data(voltage2.key, line)
    print(line)
    numero = numero +1
    time.sleep(1)

# close the pointer to that file
```

En este ejemplo asignaremos dos fuentes de datos creados en el panel llamadas voltage-modulo-1 y voltage-modulo-1, en las que mostraremos los datos leidos del sensor.



.

Figura 17: Ejemplo de panel(dashboard) utilizado.

7

Conclusiones y líneas futuras

7.1 Conclusiones

El subsistema Industrial I/O del Kernel de Linux permite construir drivers más estándar y con una mayor robustez al facilitar el trabajo de comunicación entre el área del Kernel y el espacio de usuario, así como proporcionar facilidades de programación del dispositivo. Al estructurar la entrada y salida en canales de datos se establecen unas estructuras más fáciles de manejar y comprender tanto para el desarrollador, como para el usuario del dispositivo, evitando así fallos en la programación del driver y obteniendo controladores de mayor fiabilidad, además de conseguir un modelo más estándar para el desarrollo de estos controladores.

Este TFG nos ofrece una visión general de la programación de un controlador en el Kernel de Linux, el cual nos enseña la dificultad y la especialización necesaria para estos desarrollos. Cada dispositivo hardware tiene sus características y sus especificaciones técnicas de funcionamiento, lo cual requiere unificar muchos de los conocimientos adquiridos en diferentes asignaturas de esta titulación como física, matemáticas, dispositivos electrónicos. Por tanto, cada desarrollo de un nuevo controlador comparte las estructuras de funcionamiento, pero se debe conocer los detalles a nivel hardware, esto requiere una gran especialización y conocimiento de los dispositivos.

En cuanto al desarrollo de este TFG considero que se han cumplido todas las expectativas en cuanto al desarrollo del controlador IIO, incluso mejorando lo que iba a ser este trabajo al añadir el servicio de Adafruit.io para la visualización de datos a forma de demostración. El driver proporciona una funcionalidad básica que puede ser mejorada en líneas futuras.

7.2 Líneas futuras

En líneas futuras que han quedado por desarrollar en este TFG:

- ✓ Crear una librería más completa para manejar el dispositivo con interfaz para cambiar los parámetros de configuración del dispositivo INA219, definir el rango entre 16, 32 V, definir los posibles valores de ganancia entre 1 y 8. Definir el número de bits para usar en convertidor de analógico a digital (valor máximo 12bits). Esto afecta a la velocidad de conversión y a la sensibilidad.

- ✓ Implementar más canales, para mostrar más información del dispositivo y en diferentes formatos.

- ✓ Usar el driver para monitorizar otros sistemas como el consumo de baterías.

- ✓ Implementar un circuito con el componente ina219 en vez de usa una placa de desarrollo.

Referencias

1. Linux Device Drivers Developer By: John Madieu. ISBN 978-1-78528-000-9

2. Datasheet sensor: http://www.ti.com/lit/ds/symlink/ina219.pdf

3. Linux Device Drivers (2005) By: Jonathan Corbet; Alessandro Rubini; Greg Kroah-Hartman. ISBN-13: 978-0-596-00590-0

4. Interfaz i2c en el kernel de Linux: https://www.kernel.org/doc/html/v4.12/driver-api/i2c.html

5. Device Tree: https://www.devicetree.org/specifications/

6. Datasheet módulo: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ina219-current-sensor-breakout.pdf

7. Librería IO para Raspberry Pi de Adrafruit gratuida: https://io.adafruit.com/

8. Driver linux para pantalla oled ssd1306 i2c

   https://riuma.uma.es/xmlui/handle/10630/16907

Apéndice A

Manual de Instalación

Requerimientos:

- ✓ Raspberry Pi 3
- ✓ Tarjeta SD al menos 8GB
- ✓ Raspberry Pi GPIO Extension Board
- ✓ 4 sensores INA219
- ✓ Pantalla oled ssd1306
- ✓ Diodos leds
- ✓ 2 pontenciómetros
- ✓ 2 placas de desarrollo (Protoboard)
- ✓ Cables de conexión
- ✓ Soldador.

1. Instalación del sistema operativo rasphian.

El primer paso para preparar el entorno es la instalación del sistema operativo de la Raspberry Pi en la tarjeta microSD. En este TFG se ha utilizado una versión de Rasphian Stretch Lite que es una versión básica sin escritorio.

Se puede instalar el sistema operativo desde en la tarjeta SD desde diferentes entorno, en este caso se ha realizado desde un portátil con Windows y el software balenaEtcher. Para este caso, se realiza los siguientes pasos:

- ✓ Descargar la imagen de Rasphian Strecth Lite
- ✓ Descargar e instalar el software de balenaEtcher https://www.balena.io/etcher/
- ✓ Insertar la tarjeta microSD en el equipo.
- ✓ Ejecutar el software y seguir los tres sencillos pasos. Seleccionar la imagen, seleccionar la unidad donde está la tarjeta microSD y pulsar sobre el botón Flash!.

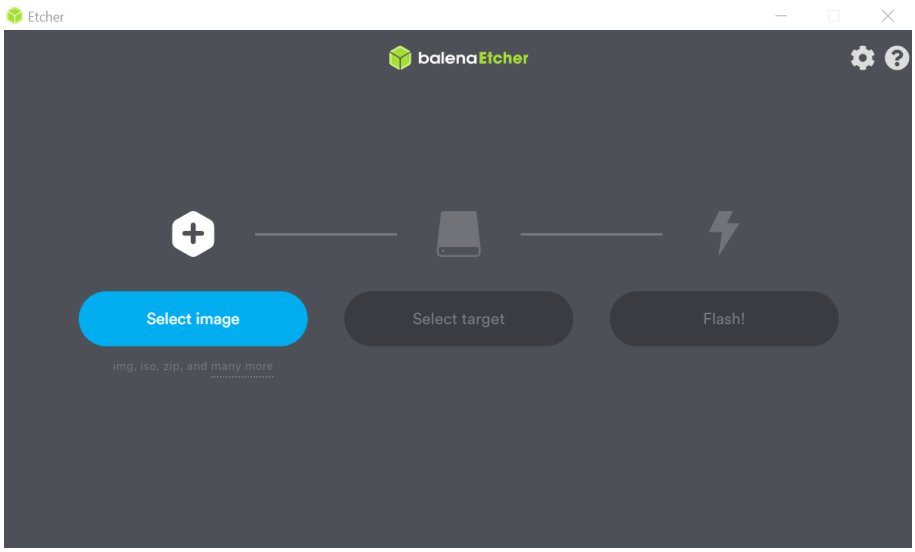


Figura 17: Interfaz del balenaEtcher.

Una vez finalizado el proceso de copia ya disponemos del sistema operativo instalado en la microSD. Si no disponemos de un teclado adicional y pantalla para la Raspberry Pi en este momento podemos activar el acceso mediante ssh. Por defecto el acceso ssh viene desactivado, para habilitarlo se crea un fichero llamado "ssh" en la partición de arranque /boot.

El usuario por defecto que viene en el sistema es pi y la password "raspberry". Para añadir un nuevo usuario utilizaremos el comando:

sudo adduser agus
sudo usermod agus –a –G pi,adm,dialout,cdrom,sudo,audio,video,plugdev,games, users,input,netdev,spi,i2c,gpio

Una vez accedido por ssh a la raspberry Pi se utiliza el comando sudo raspi-config para configurar muchas de las opciones del sistema como el locale (es-ES UTF-8) y la zona horaria (Madrid).

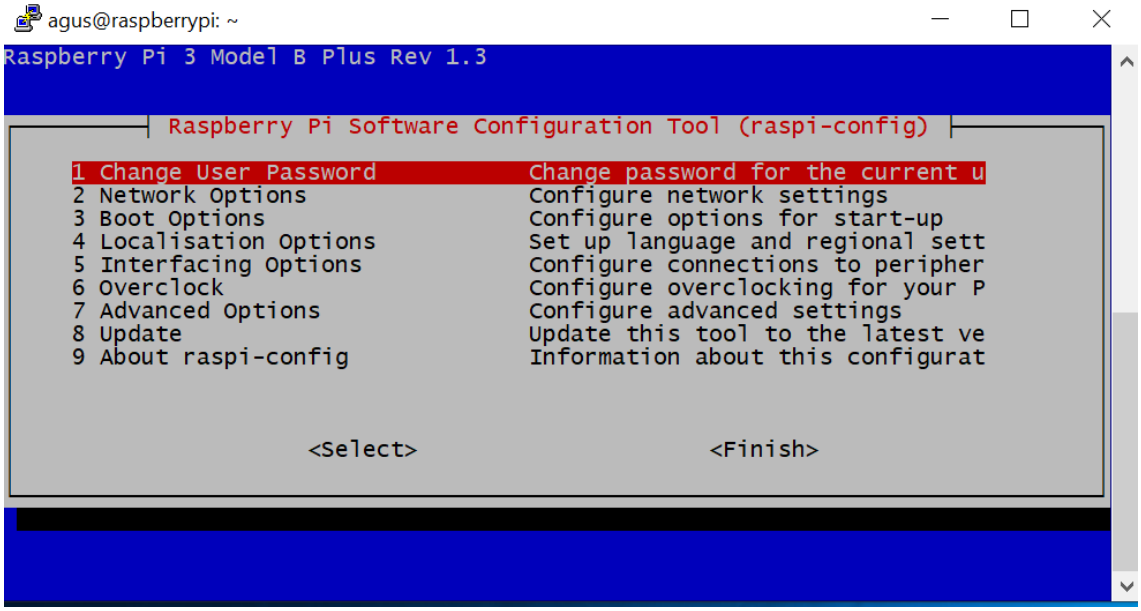


Figura 18: Raspberry Pi Software Configuation Tool (raspi-config)

Hay que activar la carga automática del módulo del kernel para I2C. Para ello, entra en la opción "5 Interface Options", una vez dentro se selecciona la opción "8 Advanced options" y luego I2C y Enable. Reiniciamos el sistema.

Se instalarán las herramientas para i2c con los comandos:

sudo apt-get install -y python-smbus
sudo apt-get install -y i2c-tools

Estas herramientas de i2c nos permiten ver toda la información de los buses I2C. Para ver la información del bus I2C-1, el bus de la Raspberry Pi donde conectaremos los dispositivos, se utiliza el siguiente comando:

```
sudo i2cdetect –y 1
```

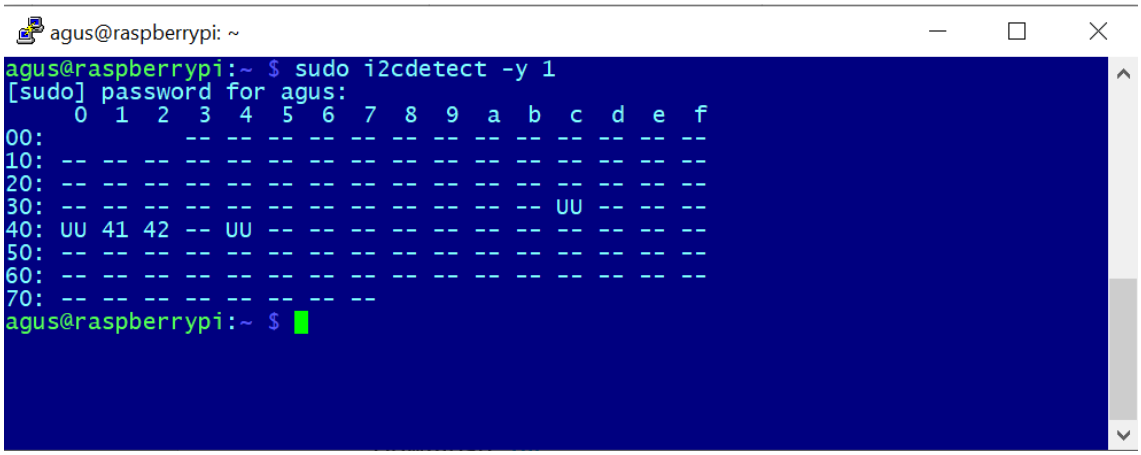


Figura 19: Información de bus I2C-1

En la figura 15 podemos ver la información del bus i2c-1 donde aparece un dispositivo en la dirección 3c (modulo para pantalla oled ssd1306), 4 dispositivos ina219 en las direcciones 40,41,42,44. Los dispositivos presentes en el bus que están en uso se indican con UU, mientras los que están presentes y no están en uso se indican con el número de la dirección (41,42). Si no detecta ningún dispositivo en la dirección se muestra cómo --.

Esta información corresponde al bus i2c-1 de la raspberry Pi en la cual se ha conectado los elementos que aparecen en la figura 16.

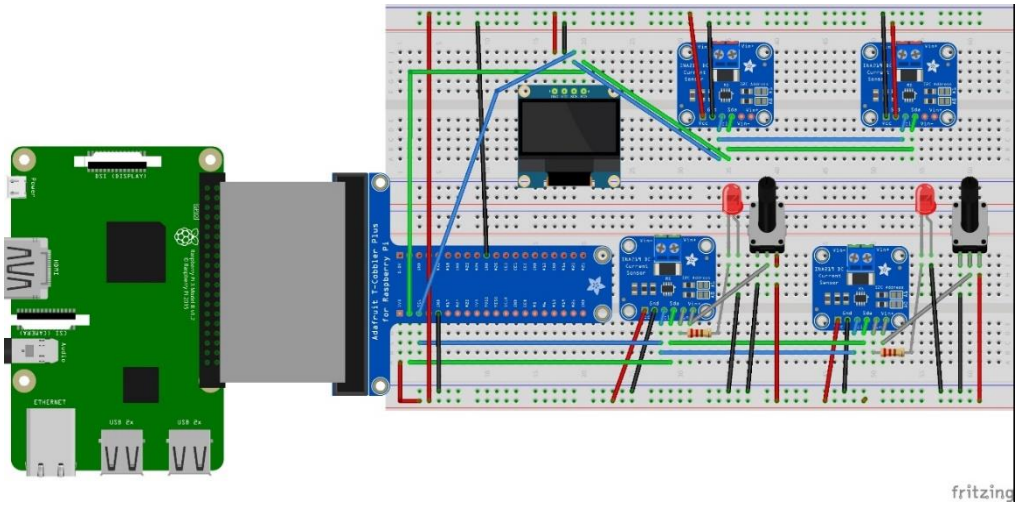


Figura 20: Esquema de conexión protoboard con la raspberry pi

Apéndice B

La Pantalla oled SSD1306

En este TFG se ha utilizado un módulo cargable para la pantalla oled ssd1306 basado el driver i2c del TFG de José Carlos Navarro González [8]. El cuál me ha ayudado a comprender muchos de los conceptos de los módulos cargables de Linux, así como la utilización y el desarrollo de controladores i2c.

La raspberry Pi no incorpora pantalla, su manejo principalmente se realiza a través de una consola ssh. Por tanto, supone de gran utilidad disponer de este dispositivo para mostrar algunos datos como la ip de la raspberry durante el proceso de arranque para poder realizar la conexión ssh.

Al módulo cargable inicial, se han incorporado muchas de las líneas futuras no implementadas, a través de la función Ioctl. La funcion Ioctl es una funcion comodín que nos va a permitir añadir funciones extra, diferentes a las típicas que podemos hacer con un fichero como leer, escribir, abrir o cerrar el fichero. En el módulo utilizado se han añadido las funciones:

- Modo inverso
- Rotar izquierda
- Rotar derecha
- Cambiar brillo de la pantalla.
- Cambiar la fuente de texto.

Para implementar estas funciones la funcion ioctrl nos permite pasar parámetros y darle otra funcionalidad a nuestro dispositivo de caracteres.

La estructura la funcion es la siguiente:

int oled___ioctl (struct inode *pinodo, struct file *pfile, unsigned int comando, unsigned long args)
Como se puede observar los argumentos de entrada a *ioctl()* son el *inodo* del dispositivo, la estructura *file* del dispositivo asociada al proceso que la invocó, un argumento numérico que identificará cual de todas las posibles acciones se quiere realizar (modo inverso, borrar pantalla, establecer brillo...) y por último otro argumento numérico (en realidad un puntero) en el que irán los posibles argumentos de las diferentes acciones a realizar, en el caso de que fuesen necesarios.

El identificador numérico del comando se construye concatenando cuatro campos de bits en un *unsigned int*:

- **type**: El número mágico del dispositivo, es una especie de clave para comprobar que el proceso que llama a *ioctl* conoce realmente cual es el *comando* para cada acción asociada al dispositivo. Simplemente se escoge uno al azar.
- **number**: el número que identifica a cada acción.

- **direcction**: En el caso de que una acción en particular realice transferencia de datos, este campo nos indica el sentido de la transferencia desde el punto de vista de la aplicación. Este campo puede tener uno de los siguientes valores:
    - **__IOC__NONE**: No habrá transferencia de datos.
    - **__IOC__READ**: Se leerán datos del dispositivo.
    - **__IOC__WRITE**: Se mandarán datos al dispositivo.
    - **size**: El tamaño de los datos transferidos, si procede, en unidades del valor de la macro **__IOC__SIZEBITS**.

Para la construcción de cada *comando*, la librería **asm/ioctl.h** nos proporciona las siguientes macros:

- **__IO (type, nr)**: para cuando no hay transferencia de datos.
- **__IOR (type, nr, size)**: para cuando se quieren obtener datos.
- **__IOW (type, nr, size)**: para cuando se quieren mandar datos.
- **__IOWR (type, nr, size)**: para cuando se quieren mandar y obtener datos.

En *asm/ioctl.h* también se definen unas cuantas macros para decodificar el comando:

- **__IOC__TYPE (nr)**: Nos da el *type* a partir del *comando*.
- **__IOC__NR (nr)**: Nos da el *number* a partir del *comando*.

- **_IOC_DIR (nr)**: Nos da el *direcction* a partir del *comando*.
- **_IOC_SIZE (nr)**: Nos da el *size* a partir del *comando*.
-

Todas estas funciones las probaremos desde una librería donde pondremos mandar al fichero los códigos de las funciones ioctrl.

```
static long ssd1306_ioctl(struct file *file, unsigned int
cmd,
                                    unsigned long args){
          int value;
          int i;

          if (_IOC_TYPE(cmd) != SSD1306_NUM_MAG){
               return -EINVAL;
        }

        if (_IOC_NR(cmd) > SSD1306_NUM_MAX){
               return -EINVAL;
          }

     switch(cmd){

          case SSD1306_BORRAR:
               printk(KERN_INFO "(ioctl %d) borrada
pantalla\n",
                         _IOC_NR(cmd));

               xpos=0;
               ypos=0;
               clear_buffer();
               display();
               break;


          case SSD1306_ROTAR_IZQUIERDA:
               printk(KERN_INFO "(ioctl %d) rotar
izquierda\n",
                         _IOC_NR(cmd));
               ssd1306_command(SSD1306_DEACTIVATE_SCROLL);
               scroll_izquierda();
          break;

          case SSD1306_ROTAR_DERECHA:
               printk(KERN_INFO "(ioctl %d) rotar
derecha\n",
                         _IOC_NR(cmd));
               ssd1306_command(SSD1306_DEACTIVATE_SCROLL);
               scroll_derecha();
          break;

          case SSD1306_ROTAR_ABAJO:
```

```
                printk(KERN_INFO "(ioctl %d) rotar
arriba\n",
                          _IOC_NR(cmd));
                ssd1306_command(SSD1306_DEACTIVATE_SCROLL);
                scroll_arriba();
          break;

          case SSD1306_ROTAR_ARRIBA:
                printk(KERN_INFO "(ioctl %d) rotar
abajo\n",
                          _IOC_NR(cmd));
                ssd1306_command(SSD1306_DEACTIVATE_SCROLL);
                scroll_abajo();
          break;
          …
```

Todas estas funciones se utilizará desde una librería donde pondremos mandar al fichero los códigos de las funciones ioctrl.

Implementación del modo inverso

En el modo inverso se ilumina los pixeles a cero en vez de uno.

```
void oled_modo (int* m) {

int res, fp;

fp = open ("/dev/ssd1306_oled", O_RDWR);

if (fp == -1) {
printf ("Error no existe el fichero que se desea escribir\n");
exit (1);
} else {
res = ioctl (fp, MODO, m);
if (res! = 0) {
printf ("Error al estable modo \n");
}
}
res = close(fp);

if (fp == -1) {
exit (1);
}
```

}

Los modos son:

0 modo normal
1 modo inverso.

Control de brillo

El control del brillo se establece entre 0 y 255.
A simple vista solo se aprecia claramente cuando cambios en 5 niveles de brillo por ejemplo 0, 50, 100, 150, 200.

```
void oled_contraste (int* c) {

int res, fp;

fp = open ("/dev/ssd1306_oled", O_RDWR);

if (fp == -1) {
printf ("Error no existe el fichero que se desea escribir \n");
exit (1);
} else {
res = ioctl (fp, CONTRASTE, c);
if (res! = 0) {
printf ("Error al rotar arriba \n");
}
}
res = close(fp);

if (fp == -1){
exit (1);
}

}
```

Scroll izquierda y derecha

```
void oled_rotaizq () {
int res, fp;
fp = open ("/dev/ssd1306_oled", O_RDWR);
if (fp == -1) {
printf ("Error no existe el fichero que se desea escribir \n");
exit (1);
} else {
res = ioctl (fp, R_IZQUIERDA);
if (res! = 0){
printf ("Error al rotar a la izquierda \n");
}
}
res = close(fp);

if (fp == -1) {
exit (1);
}
}
```

R_IZQUIERDA ejecutara los comandos incluidos en la funcion scroll_izquierda de nuestro driver:

```
static void scroll_izquierda(void)
{
// consultar datasheet para ver parámetros del comando
ssd1306_command (SSD1306_DEACTIVATE_SCROLL);
ssd1306_command (SSD1306_INVERTDISPLAY);
ssd1306_command (SSD1306_LEFT_HORIZONTAL_SCROLL);
ssd1306_command(0x00); // dummy
ssd1306_command(0x00); // inicio
ssd1306_command(0x06); // velocidad
ssd1306_command(0x0f); // fin
ssd1306_command(0x00); // dummy
ssd1306_command(0xff); // dummy
ssd1306_command (SSD1306_ACTIVATE_SCROLL);
}
```

Cambio de fuente.

Se han creado dos nuevas fuentes adicionales, con la cual disponemos de tres fuentes diferentes para escribir texto en nuestra pantalla. Una fuente Italy y otra Thin.

Ambas fuentes se han creado partiendo de fuentes 8x8 y utilizando el software 8x8 Píxel Font Editor se han rotado a la derecha para poder utilizar directamente en nuestra pantalla.

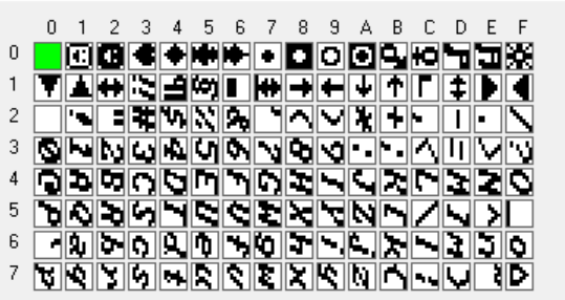


Figura 21: Fuente Italy 8x8 fuente rotada a la derecha.

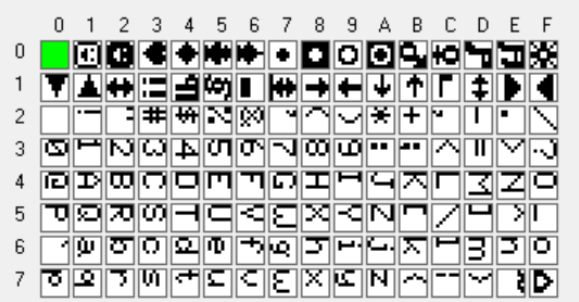


Figura 22: Fuente Thin 8x8 fuente rotada a la derecha.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga